are connected to different trees. Theorem 39.2.17: Any two vertices are connected.

Theorem 39.2.18: For a connected graph, the edge is connected if and only if the vertices are connected.

Two standard representations:

1. Adjacency matrix: A square matrix with 1's at position (i, j) if there is an edge from vertex i to vertex j.
2. Adjacency list: A linked list for each vertex containing the vertices it is connected to.

Representations of graphs: G = (V, E)

Outline

- Algorithms
- Data Structures
- Graph Algorithms

Textbook: Chapter 3, Sections 2.2, 2.3, and 2.4

CSC 310: Data Structures and Algorithms

Conference Room 104
Burling A. Cherry Hall

www.cs.umn.edu/courses/310

Elementary Graph Algorithms

[Diagram of a graph with vertices and edges labeled]
Adjacency-list vs. Adjacency matrix

- Requires $O(V+E)$ space, independent of $E$.
- Undirected graph: $A = A^T$, we can save space below diagonal.
- Weighted graph: $A[u,v]$ does not exist; $A[u,v] = 0$, $\infty$, depending on application.

Adjacency-matrix representation:

- Additional space cost.
- When graphs are weighted, matrix can store weight at no.
- Adjacency matrix preferable when graphs are small.
- When graphs are weighted, matrix can store weight at no.

BFS: How it works:

1. Visit a vertex $s$ to be the starting point.
2. Initialize distance from $s$ to all reachable nodes.
3. Enqueue node $s$.
4. Dequeue every vertex from queue $s$ to discover.
5. Determine edges of $G$.
6. Decrease $G = (V,E)$ and so on (vertex $s$ is done).
BFS: pseudocode

1. For each vertex \( v \in V(G) \) do
2. \( d[v] = \infty \)
3. color[\( v \)] = white
4. \( Q \leftarrow \{ v \} \)
5. while \( Q \neq \emptyset \) do
6. \( u = \text{Dequeue}(Q) \)
7. for each vertex \( v \in \text{adj}(u) \) do
8. if \( \text{color}[v] = \text{white} \) then
9. \( \text{color}[v] = \text{grey} \)
10. \( d[v] = d[u] + 1 \)
11. \( Q \leftarrow \text{Quick Enqueue}(Q, v) \)
12. if \( \text{color}[u] = \text{grey} \) then
13. for each vertex \( v \in \text{adj}(u) \) do
14. if \( \text{color}[v] = \text{white} \) then
15. \( \text{color}[v] = \text{grey} \)
16. \( d[v] = d[u] + 1 \)
17. \( Q \leftarrow \text{Quick Enqueue}(Q, v) \)
18. end while

BFS: algorithm

1. For each vertex \( v \in V(G) \) do
2. \( d[v] = \infty \)
3. color[\( v \)] = white
4. for each vertex \( u \in V(G) \) do
5. if \( \text{color}[u] = \text{white} \) then
6. \( d[u] = 0 \)
7. \( \text{color}[u] = \text{grey} \)
8. repeat
9. \( \text{color} \leftarrow \text{grey} \)
10. while \( \text{color}[u] = \text{grey} \) do
11. for each vertex \( v \in \text{adj}(u) \) do
12. if \( \text{color}[v] = \text{white} \) then
13. \( \text{color}[v] = \text{grey} \)
14. \( d[v] = d[u] + 1 \)
15. \( Q \leftarrow \text{Quick Enqueue}(Q, v) \)
16. while \( Q \neq \emptyset \) do
17. \( u = \text{Dequeue}(Q) \)
18. for each vertex \( v \in \text{adj}(u) \) do
19. if \( \text{color}[v] = \text{white} \) then
20. \( \text{color}[v] = \text{grey} \)
21. \( d[v] = d[u] + 1 \)
22. \( Q \leftarrow \text{Quick Enqueue}(Q, v) \)
23. end while
24. end if
25. end while
26. end for
27. end if
28. end for
29. end repeat
30. end if
31. end for
32. end for
33. end algorithm
BFS's Algorithm

Theorem 23.4: Correctness of BFS

Let the shortest path from s to v be followed by the edge (e[a][a]),
where e[a][a] = 0, or the shortest path from s to a

Union of terminations, p(x) = [a], so every vertex except the source s

The breadth-first tree of G is the breadth-first search graph.

Lemma 23.2: Define BFS's successor

To find the successor of a vertex, we must look at how the predecessor

Lemma 23.3: When BFS is run on an undirected graph G,

Lemma 23.4: When BFS is run on a directed graph G,

BFS's Algorithm

Path: Each vertex is on exactly one path from the root. There is no backtracking.

Importance of Thorough BFS's Correctness:
A path from the root to a vertex is a shortest path from the root.
Depth-first Search: choose a root and exp...
When an directed graph is not acyclic, a topological sort does not exist. So that all directed edges go from left to right.

A topological sort is an ordering of the nodes along a horizontal line in a node so that no node is left to another node in a such that there is any directed edge from a node to a node in the ordering. A topological sort of a digraph $G = (V, E)$ is a linear ordering of all

**Topological sort**

### Example

![Graph Example](image)

**DFS**

1. Mark the head of vertices from a node that has no incoming edges.
2. Call DFS $(G)$ to compute $f(v)$ for each vertex $v$.
3. Topologically sort $(G)$.

**BFS**

1. Compute $d(v)$ for each vertex $v$.
2. One possible order of getting done: topological sort.